# Heap SnapshotS

### References

#### Heap Concepts

* [terminology](https://developer.chrome.com/docs/devtools/memory-problems/memory-101/)

#### Jerryscript

* Fitbit memory-profiler structures: developer-bridge\node\_modules\@fitbit\memory-profiler\src\types.ts
* [Jerryscript heap profiler?](https://yodaos-project.github.io/ShadowNode/devs/Optimization-Tips.html)
* [Jerryscript heap snapshot PR](https://github.com/jerryscript-project/jerryscript/pull/2605)

#### V8

* [V8 heap structure](https://blog.dashlane.com/how-is-data-stored-in-v8-js-engine-memory/)
* [V8 HeapSnapshot Class Ref](https://v8docs.nodesource.com/node-0.8/d8/deb/classv8_1_1_heap_snapshot.html)

#### Firefox Snaps

* [snap format](https://searchfox.org/mozilla-central/source/devtools/shared/heapsnapshot/CoreDump.proto)
* [repo to query snaps](https://github.com/jimblandy/fxsnapshot)

#### Source Mapping

* [JS](https://indepth.dev/posts/1230/source-maps-from-top-to-bottom)
* [Google spec](https://docs.google.com/document/d/1U1RGAehQwRypUTovF1KRlpiOFze0b-_2gc6fAH0KY0k/edit)
* [repo to decypher](https://github.com/mozilla/source-map/)
* [Visualisation tool](https://evanw.github.io/source-map-visualization/)

#### JS Trees

* [stackoverflow](https://stackoverflow.com/questions/5636375/how-to-create-a-collapsing-tree-table-in-html-css-js)
* [examples](https://onaircode.com/html-css-tree-view-examples/)
* [popover](https://popper.js.org/)

### Developer-Bridge

#### Installation

install (update) node-gyp: npm install node-gyp

install (update) keytar: npm install keytar

yarn: yarn install

yarn build

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

install node 14

install yarn

install git

in git Bash\*, yarn install

in git Bash\*, yarn build

cd packages/memory-profiler/lib

node cli.js

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

transferred whole of developer-bridge from node 14 computer to node 16 computer

in git Bash\*, yarn install

in git Bash\*, yarn build

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* powershell or WSL might have worked; not tried.

Having been built in a particular directory, the package can’t be moved to a different directory and executed without rebuilding(?). [Mine requires d:\Temp\developer-bridge\...]

#### Usage

Get binary representation of memory heap:

* Run clockface/app.
* When clockface/app is executing code of interest (eg, is within a function suspected of high memory usage), use CLI command heap-snapshot.

Convert binary to JSON:

* cd packages/memory-profiler/lib
* node cli.js [args\*]

\* heap-snapshot creates filenames with a space in them. Remove the space for an easier life.

### Source Mapping

#### Example

| **source index.js** | **app.fba index.js** | **index.js.map** | **Snapshot** |
| --- | --- | --- | --- |
| myNumberVariable | — (optimised out) | 4 | — |
| myStringVariable | o | 2 | node "repr":"o" |
| myArrayVariable | r | 5 | edge “name”:”r” |
| myLocalVariable | e | 7 | node "repr":"e" |
| myNumberSumVariable | l | 8 | node "repr":"l" |

#### How to Disable

Liam: “Removing minify will stop symbol names getting mangled in most cases. You can literally comment out one line in compile.ts in toolchain to try it.”

Aaron: “I'm assuming it's here we'd comment out compression: <https://github.com/Fitbit/fitbit-sdk-toolchain/blob/master/src/compile.ts#L139>. The helpful comment above that line (kuddos!) https://github.com/mishoo/UglifyJS#source-maps-and-debugging suggests using mangle instead. Perhaps that's not helpful though in this scenario.”

Liam: “Comments out the whole terser block. Remove the tool from the equation. The way it works is that sourcemaps get combined, if you don't call the tool at all there's no sourcemap to combine.” [Is it possible that the snapshot could contain additional info if the sourcemap is retained; *ie*, if terser is used?]

See “node\_modules\@fitbit\sdk\lib\compile.js”.

Even without terser, built code still differs from source:

* The content of imported modules is included in app/index.js.
* Multiple statements on one line are placed on separate lines.
* Non-standard use of { } (and maybe other tokens) is standardised.

Without terser, heap contains:

* original symbol names in node repr fields
* original symbol names in edge name fields
* position line numbers that more closely match source (but are still larger by 1 or 2 or ?).

### Heap Snapshot Structures

#### Node Structure

All nodes have id, type, size.

Some have repr.

Some have position.

None have repr AND position.

id < 0x10000000 seem to be system nodes, or at least not explicit symbols in source code. They can be user-code array values.

First node with id>=0x1000000 and size===24 may be base node for third-party memory entities.

The node position field seems to be for functions (including anon) only (such nodes have type===Closure). position is not specified for variables. position is not used for code in global scope, so position can’t be used to identify all code.

Nodes with type===’Code’:

* size field may be correct and useful (but the node might retain other nodes with non-zero size: unchecked).
* One ‘from’ (parent) node may have type===Closure and contain a position field.
* ‘to’ (child) nodes may have one of type===’Sourcemap’ and another with repr indicating filename.

“Retained size” = self + sizes of the objects that are reachable ***only from this object***.

#### Edge Structure

All edges have type, to, from.

Some have name. name may correspond to source symbol identifier name (minified if terser used).

Most nodes seem to be leaf nodes (ie, have no edge.from edges).

#### Array Representation

Array elements may be ‘pooled’ into groups of nodes (maybe two elements per group), and each such group may be linked from the array’s parent node. This way, an array of 1000 elements may only have, say, 501 direct children.

### Test Case

Object

Array with values != indexes

Function

anon function (eg, event handler)

imported module with its own symbols

same local var name used in different functions

Ensure code is RUNNING when snapshot taken

Initially, avoid calling functions repeatedly so the heap doesn’t contain multiple instances of local variables that can be GCed.

no terser

### Retained Size

“Retained size”: objects that are reachable ***only from this object***.

Hypothesis: objects that are reachable only from this object must have from.length===1.

What about a grandchild of O with two parents, but both of those parents are only reachable from O? See diagram. Grandchild should presumably be retained by its grandparent, but not by either of its parents.

#### How to search node n

For each child c of n:

* For each parent p of c:
* If p is NOT retained, neither is c.
* If ALL parents are retained:
* Mark c as retained
* Add c to search queue. Use breadth-first search so that parents are processed before children.

While search queue isn’t empty:

* Take first node n from queue. We know n itself is retained; ie, all of its parents are retained.
* Repeat from top.

### Goals

Given a code entity (eg, function or variable name), be able to identify the corresponding heap nodes.

Given a heap node, be able to identify the corresponding code entity (if any).

Be able to answer the following questions:

* How much memory is a specified node retaining?
* What nodes are being retained by a specified node?
* What nodes are retaining a specified node (*ie*, what nodes would have to be released in order to release the specified node)?
* Which nodes are eligible for GC, and which nodes must be retained? (There can be multiple instances of a single variable in the heap, corresponding to multiple calls of the function that contains it. If execution is within the function when the snapshot is taken, all but one instance should be GCable; if execution is outside the function, every instance should be GCable.)

### Questions

Why do many nodes have size===0?

Why do some heap.json have position fields, and others don’t have any? (No functions in the latter?)

Why do some heap.json have repr fields that contain (minified) symbol names, and others don’t?

Why are most nodes unconnected (ie, having no from or to edges)?

Graph seems to be recursive (*ie*, cyclic). How to calculate total memory size associated with a node? (Assume we count each node only once, and ignore revisits.)

Why do functions sometimes have more than one corresponding Closure node? (Consequence of multiple calls, resulting in GCable instances?)

Some edges[] are identical (*ie*, duplicated). What does this indicate?

How can we trace from a node that indicates a var’s name to the node(s) that represent its value? Is it possible that some missing links are contained within data structures other than this heap?

## Scratch Area

### How to Relate Var Name to Var Value?

#### myStringVariable

Nodes and Edges with Helpful Field Values:

* node id=279546217 type=String size=24 from=1 to=0 repr="myStringVariable"
* node id=279545305 type=String size=32 from=1 to=0 repr="myStringVariableValue"
* edge type=property from=279545731 to=279545305 name=myStringVariable

Relationships:

* The variable name (279546217) has an edge from 279545731.
* The variable value (279545305) has an edge from 279545731.
* 279545731 is therefore a common ‘from’ node (ie, parent?) for both var name and value.
* Reversing this, given the common ‘from’ node, can we find both the var name and value?
* 279545731 contains ‘to’ nodes for other vars (myObject), not just myStringVar :(
* Given var name node, can we trace back to its ‘from’, then forward through its ‘to’ to obtain value node(s)?
* Given var value node, can we trace back to its ‘from’, then forward through its ‘to’ to obtain name node(s)?

#### myNumberVariable

Nodes and Edges with Helpful Field Values:

* node id=279546009 type=String size=24 from=1 to=0 repr="myNumberVariable"
* 279546009 has ‘from’ edge with 279545715.
* 279545715 to ‘to’ edges with 279546009 (myNumberVariable), 279545937 (myExportedStringVariable) and 279545273 (myExportedStringValue). :(

### How to Test Retained Node Counting without double-Counting

Identify a source entity node (mySharedVariable) that is retained by two other source entity nodes (innerFunction1 and innerFunction2).

Check that mySharedVariable node is retained by outerFunction node and isn’t counted twice.

#### mySharedVariable

#### innerFunction1

### HTML GUI for Node Exporation

Start with a specified initial node.

Indicate edges to and from node.

Allow user to expand (explore) to or from nodes. When done, a list of linked nodes should be displayed. Use mouse hover to provide full details of nodes and edges.

Pop-up right-click context menu: calc retained size, etc.

Limit number of nodes to open, or warn if many?

Newly-opened nodes should have the same appearance as the initial node, since they support the same features (hover, menu, drill to/from).

Use indentation and maybe colour to distinguish ‘to’ and ‘from’ edges.

Mark previously-visited nodes so that it’s obvious that recursion happens here. Perhaps hyperlink to previous instance.

Could use HTML canvas.

#### Example

Initial node:

(from count) — N — (to count)

Click on (to count):

(from count) — N —

(from) — Nto1 — (to)

(from) — Nto2 — (to)

(from) — Nto3 — (to)

(from) — Nto4 — (to)

Click on Nto2’s (from):

(from count) — N —

(from) — Nto1 — (to)

— Nto2 — (to)

(from) — N2a — (to)

(from) — N2b — (to)

(from) — N2c — (to)

(from) — Nto3 — (to)

(from) — Nto4 — (to)

Click on N2b’s (from):

(from count) — N —

(from) — Nto1 — (to)

— Nto2 — (to)

(from) — N2a — (to)

(from) — N2b — (to)

(from) — N2d — (to) // needs to go left further

(from) — N2c — (to)

(from) — Nto3 — (to)

(from) — Nto4 — (to)

Issues:

* Continually expanding ‘from’ edges can require display of nodes to left of original node.
* It’s unclear that Nto3 is a child of N when Nto2 has been expanded.
* What does it look like if expanding both from and to?
* Consider always expanding nodes to right side, regardless of whether ‘to’ or ‘from’.